Web Scraping with Python

Learning Outcomes

By the end of this topic, you will have achieved the following learning outcomes:

- I can **define** web scraping as used in collecting data.
- I can **describe** the importance of web scraping.
- I can **describe** the limitations of web scraping.
- I can **use** various python tools to perform web scraping.

"If you want to understand people, especially your customers, then you have to be able to possess a strong capability to analyze text." Paul Hofmann

Reading

What is Web Scraping?

Web scraping is the process of extracting data from the website.

This process transforms unstructured data found on the web into well-structured, machine-readable data that can be used for data analysis. It may entail downloading the content of a website and then extracting the data we would need.

Below are a few concepts that you should know about websites before going further into web scraping.

- A collection of web pages makes up a website. A webpage can be an HTML or XHTML document from which we will be extracting data
- **HTML** and **XHTML** are the languages in which web pages are written. Web pages contain source code which is made up of tags i.e. <h1>, <h2>, <p>, <a>, , etc. which in turn contain data. Tags can also contain classes

which can be useful in referencing i.e. <h1 class="main_title">, in this case the h1 tag has a class "main_title"..

There are some websites that provide a way of conveniently accessing data via an API (Application Programming Interface), however, many websites don't offer such alternative options. Hence, the use of web scraping techniques to retrieve data from websites.

Why is Web scraping useful?

We might consider web scraping if we're performing the following:

1. **Innovation:** Web scraping enables businesses to consolidate information from various sources in order to create new products and innovate faster.
2. **Market Research:** We can scrape data i.e. online reviews, to better understand customers.
3. **Building Large Databases:** Web scraping helps to further build and enrich existing datasets with further related data.
4. **Automation:** Web scraping helps automate the process of manually collecting/copy-pasting data from the web.
5. **Affordable and Accurate:** Automation achieves the goal of collecting the data in an efficient, accurate, and budget-friendly manner.

Before working on any web scraping project, we need to perform several checks. These checks can be viewed as requirements that would contribute to a successful web scraping project:

- Ensuring that the information gathered is worth the effort to build a web scraper.
- Downloading information that can be legally and ethically gathered by a web scraper. We can ask questions like:
    - Am I scraping copyrighted material?
    - Will my scraping activity compromise individual privacy?
    - Am I making a large number of requests that may overload or damage a server?
    - Is it possible the scraping will expose intellectual property I do not own?
    - Are there terms of service governing the use of the website, and am I following those?
- Have some knowledge on how to find the target information on the target website.
- Have the right tools installed in our environment i.e. Pandas, Beautiful Soup, and Selenium.
- Be able to use the mentioned tools to source and manipulate the data.

How is Web Scraping done?

In a summary, when we refer to the act of performing web scraping, we refer to the act of writing code that sends a request to the server where the website is hosted. The server then returns back the HTML/XHTML document. We then parse through the document and specify which tag's content we would want to get/scrape the data.

This process can be broken down into the following steps:

1. **Visual inspection: Figuring out what to extract**
   This process would entail using a browser to locate where within the webpage data can be extracted from. It involves locating the specific tags where the data is contained in. We can achieve this by right-clicking anywhere on the webpage and going to “inspect” and selecting the element that we want to inspect.

2. **Making an HTTP request to the webpage**
   During this step, a request is made to the server to retrieve the webpage which would be in the form of an HTML document or XHTML document.

3. **Parsing the HTML**
   This step involves getting the data from the HTML document by specifying the specific tag where the data is located. This is done once we have located the specific element in which we would want to get data, then navigating to that section.

4. **Persisting/ Utilizing the relevant data**
   Data can then be compiled and stored in a table for further analysis i.e. topic summarization, classification, basic data analysis, etc. In many cases, while performing this step, we might also create a nested loop in our web scraper, for us to iterate through the item i.e. list, that contains our desired data.

Limitations of Web Scraping

- **Asynchronous loading and client-side rendering:**
  - Some websites are made up of Javascript Frameworks such as ReactJs, AngularJS, VueJS, etc. which may not be easy to scrape as the structure of the webpage might not contain the information that we would expect as per visual inspection. Examples of websites with the use of such JavaScript Frameworks include Twitter, Facebook, and webpages with preloaders like loading spinners. In such a case, the Selenium library can be used as an alternative to Beautifulsoup.

- **Handling Authentication:**

    - Many websites have some form of authentication which would need to be taken care of during scraping. To go around this, one would need to create a session that will maintain cookies and persist our login, making it possible for us to get pages that need authentication.

- **Server-side blacklisting:**
    - Some websites might have anti-scraping mechanisms set up on the server-side to analyze incoming traffic and browsing patterns, and block automated programs from browsing their site. Such websites might analyze the rate of requests, they might inspect request headers to detect non-human users, or might have a way of determining very defined patterns in the way the website is being browsed which would lead to blacklisting.

    - To tackle serverside blacklisting, one may opt to use proxy servers and IP rotation in the web scraper. In addition, random time waits between the actions i.e. making requests, can also be used which will ultimately randomize the browsing pattern and making it harder for the server to differentiate between the web scraper and a real-world user.

- **Redirects and Captchas:**
    - Some websites can filter suspicious clients and may redirect web scraping requests to pages containing quirky captchas, which a web scraper needs to solve to prove that "it's a human". Companies like Cloudflare provide anti-bot or DDoS protection services, which make it even harder for bots to make it to the actual content.

    - One can avoid captchas to some extent by using proxies and IP rotation. If these do still persist one can use API services like Death by Captcha, Antigate, and Anti Captcha.

Python Libraries for Web Scraping

The process of web scraping can be a tedious and lengthy process, however, the use of various tools provided by the python programming language makes it easy and effective to perform this process. Python libraries such as Pandas and Beautiful Soup are such libraries that make it easy to perform this process.

- **Pandas**: This is the most common library for performing data manipulation. Once data has been scrapped, it can be stored in a pandas data frame for further manipulation.

- **Beautiful Soup**: This is a python library that is used for parsing HTML/XHTML and extracting data. It helps navigate HTML/XHTML documents in order to find what we need, thus making it quick and painless to extract the data from the web pages.

- **Selenium**: This is a web browser automation library that can be used to perform actions such as clicking buttons, entering information in forms, searching for specific information on the web pages, etc.

- **Requests**: This is a python library for making HTTP requests to the server in order to fetch the HTML/XHTML documents.

- **urllib**: This is a python library for URL handling.

References

You can also use the following resources for further reading.

1. The Web Scraping Template [Link]
2. Web Scraping [Link]
3. Introduction to Python Web Scraping [Link]
4. Python Web Scraping [Link]
5. Web Scraping with Python [Link]
6. Web Scraping Tutorial [Link]
7. Python Web Scraping Practical Introduction [Link]